REMARKS

Applicant amended independent claim 1 to clarify that memory reference source instructions are converted to an intermediary format (namely, the standard formatted memory access instructions), and that it is these intermediary standard formatted memory access instructions that are matched to instructions patterns to generate matches used to obtain the resultant vector memory instructions. Support for this clarification is provided throughout the originally filed application, including, for example, at page 3, line 21, to page 4, line 11. Applicant similarly amended independent claims 15 and 27.

Additionally, applicant amended independent claim 15 for greater clarity. The specification describes these features of independent claim 15 at, for example, page 3, lines 10-31, of the originally filed application. Also applicant amended claims 4-5, 16 and 30 to make the language recited therein consistent with the amended language of the respective independent claims. After these amendments, claims 1-30 are pending in the above-identified application. Claims 1, 15 and 27 are independent.

The examiner rejected claims 1-8, 10-20 and 22-30 under 35 U.S.C. §102(b) as being anticipated by U.S. Pre-Grant Publication No. 2004/0006667 to Bik et al. The examiner further rejected claims 9 and 21 under 35 U.S.C. 103(a) as being unpatentable over Bik in view of the Microsoft Press Computer Dictionary.

Specifically, regarding independent claim 1, the examiner stated in the June 14, 2006, Final Action:

> 3. With respect to claim 1, Bik teaches a method comprising:
> converting memory access instructions in a source code into standard formatted memory access instructions, in par. 31.
> generating partitions containing the standard formatted memory access instructions, in par. 31.
> generating a match set, the match set including matches of instruction patterns to the standard formatted memory access instructions in the partitions, in par. 31; and
> transforming the matches to vector memory access instructions, in par. 31.

Applicant disagrees.

Applicant's independent claim 1, as amended, recites "converting memory access instructions in a source code into intermediary standard formatted memory access instructions; generating partitions containing the intermediary standard formatted memory access instructions; generating a match set, the match set including matches of instruction patterns to the intermediary standard formatted memory access instructions in the partitions; and transforming the matches to vector memory access instructions."

As described in the originally filed application, applicant's method involves:

> **the compiler 18 transforms (104) as many formatted memory access instructions as possible to a standard format in which the memory address reference is in the form of base address plus offset, where base address is an address that serves as a reference point for other addresses. For example, a base address can indicate the beginning of a program. The address of every instruction in the program can then be specified by adding an offset to the base address. Transforming a memory access instruction to the form of base address plus offset facilitates memory access vectorization by making it easier to combine several memory accesses together.**
>
> **After the pass of transforming memory access instructions the compiler 18 generates (106) memory access partitions for each set of memory reads or memory writes to a particular memory bank based on a data flow graph. A memory access partition contains groups of memory access instructions inside one basic block. All memory access instructions inside a group (also called subnode of a memory access partition) perform the same memory access (write or read) to the same memory bank. The compiler 18 vectorizes (108) multiple memory access instructions based on instruction pattern matching performed for each group (subnode) of each memory access partition. (page 3, line 21, to page 4, line 11 of the originally filed application)**

Applicant's method, as recited in claim 1, converts the source-code memory access instructions into intermediary instructions, and generates partitions into which the intermediary instructions are grouped. Converting the source-code memory access instructions into an intermediary standard format enables determining whether the converted instructions perform memory-access operations to nearby memory locations. The intermediary instructions are subsequently matched to instructions patterns, and the matches obtained are transformed into the resultant vector instructions.

In contrast, Bik describes a method and apparatus for implementing adjacent, non-unit stride memory access patterns using single instruction, multiple data (SIMD) instructions (Bik, page 1, paragraph 1). As Bik explains:

> **[0031] A method and apparatus for implementing adjacent, non-unit stride memory access patterns utilizing SIMD instructions are described. In one embodiment, the method includes compiler analysis of a source program to detect vectorizable loops having serial code statements that collectively perform adjacent, non-unit stride memory access. Once a vectorizable loop containing code statements that collectively perform adjacent, non-unit stride memory access in detected, the system compiler vectorizes the serial code statements of the detected loop to perform the adjacent, non-unit stride memory access utilizing SIMD instructions. As such, the compiler repeats the analysis and vectorization for each vectorizable loop within the source program code. (emphasis added, Bik, page 2, paragraph 31)**

Bik further explains that:

> **[0093] Referring again to FIG. 10, at process block 602, a system compiler analyzes a source program to detect loops having one or more serial code statements that collectively perform adjacent, non-unit stride memory access. As described above, serial code statements that collectively access adjacent elements in memory can be vectorized utilizing embodiments of the present invention. In one embodiment, the source program is first analyzed to detect each vectorizable loop with the source program.**
>
> **[0094] As described above, vectorizable loops refer to loops containing serial code statements that can be replaced with SIMD instruction statements to perform parallel processing of data elements. Accordingly, at process block 604, it is determined whether collective unit-stride memory access is detected while analyzing the source program. When the system compiler detects serial code statements that collectively perform unit-stride memory access, process block 660 is performed. At process block 660, the system compiler vectorizes serial code statements of each detected loop to perform adjacent, non-unit stride memory access, utilizing SIMD instructions ("SIMD vectorization"). (Bik, page 9, paragraphs 93-94)**

Bik's compiler detects source-code loops having code sequences that perform repeated memory access operations to or from near-adjacent memory locations. Once those source-code loops are detected, Bik's compiler vectorized those vectorizable loops into code statements that use SIMD instructions. Examples of the vectorizable source-code loops that Bik's compiler detects and then transforms into code sequences that use SIMD instruction are provided, for example, in Tables 3, 5, 7 and 9 of the Bik reference.

Thus, while Bik presumably performs some type of pattern matching to identify suitable source-code loop candidates for vectorization, the pattern matching is performed directly on the source code and not on intermediary instructions converted from the source code. Indeed, Bik's compiler has no need to convert source-code memory access instructions to intermediary

standard format memory-access instructions because Bik's compiler detects source-code loops whose corresponding memory access instructions are already known to access nearby memory locations.

Accordingly, Bik fails to disclose or suggest at least the features of "converting memory access instructions in a source code into intermediary standard formatted memory access instructions," or "generating a match set, the match set including matches of instruction patterns to the intermediary standard formatted memory access instructions in the partitions," as required by applicant's independent claim 1.

Additionally, the examiner stated in the August 30, 2006, Advisory Action:

> 3. With respect to Applicants arguments on pages 4-5 that Bik does not teach "generating partitions containing the standard formatted memory access instructions", the Examiner respectfully disagrees. As can be seen in fig. 7, the standard formatted memory access instructions (the SIMD instructions "movaps XmmO, [eax]" and "movaps Xmm1, [eax+16]") are separate partitions that when combined, achieve the store to memory array "a". (Advisory Action, page 2)

Applicant respectfully disagrees.

The examiner appears to suggest that individual instructions constitute partitions. However, as recited in applicant's independent claim 1, what is being generated are "partitions containing the intermediary standard formatted memory access instructions." Thus, the generated partitions of applicant's independent claim 1 correspond to grouping of instructions, and more particularly, to grouping of the intermediary instructions generated from the source-code. The use of these generated partitions facilitates identifying suitable memory-access instruction candidates that potentially could be combined to yield a resultant single vectorized memory-access instructions. Bik neither describes nor suggests generating groups that contain sets of memory access instructions, and Bik certainly does not describe that such partitions contain intermediary memory access instructions. Rather, as explained above, Bik does not generate intermediary instructions, but instead converts the source code directly to SIMD instructions.

Thus, Bik also fails to disclose or suggest at least the feature of

generating partitions containing the intermediary standard formatted memory access instructions," as required by applicant's independent claim 1.

Because Bik fails to disclose or suggest any of "converting memory access instructions in a source code into intermediary standard formatted memory access instructions," "generating partitions containing the intermediary standard formatted memory access instructions" and "generating a match set, the match set including matches of instruction patterns to the intermediary standard formatted memory access instructions in the partitions," applicant's independent claim 1 is therefore patentable over the cited art.

Claims 2-14 depend from independent claim 1 and are therefore patentable for at least the same reasons as independent claim 1.

Independent claims 15 recites "converting source code that includes memory access instructions that read or write less than a minimum data access unit (MDAU) to intermediary code that includes memory access instructions that read or write a multiple of the minimum data access unit; converting the memory access instructions of the intermediary code into intermediary memory access instructions that have a format including a base address plus an offset; grouping subsets of the intermediary memory access instructions into partitions; and vectorizing the intermediary memory access instructions in the subsets that match instruction patterns." For at least similar reasons as those provided with respect to independent claim 1, at least these features are not disclosed by Bik. Applicant's independent claim 15 is therefore patentable over the cited art.

Claims 16-26 depend from independent claim 15 and are therefore patentable for at least the same reasons as independent claim 15.

Independent claim 27 recites "convert memory access instructions residing in a source code into intermediary standard formatted memory access instructions; generate partitions containing the intermediary standard formatted memory access instructions; generate a match set, the match set including matches of instruction patterns to the intermediary standard formatted memory access instructions in the subsets." For reasons similar to those provided with respect to independent claim 1, at least these features are not disclosed by the cited art. Accordingly, independent claim 27 is patentable over the cited art.

Claims 28-30 depend from independent claim 27 and are therefore patentable for at least the same reasons as independent claim 27.

It is believed that all the rejections and/or objections raised by the examiner have been addressed.

In view of the foregoing, applicant respectfully submits that the application is in condition for allowance and such action is respectfully requested at the examiner's earliest convenience.

All of the dependent claims are patentable for at least the reasons for which the claims on which they depend are patentable.

Canceled claims, if any, have been canceled without prejudice or disclaimer.

Any circumstance in which the applicant has (a) addressed certain comments of the examiner does not mean that the applicant concedes other comments of the examiner, (b) made arguments for the patentability of some claims does not mean that there are not other good reasons for patentability of those claims and other claims, or (c) amended or canceled a claim does not mean that the applicant concedes any of the examiner's positions with respect to that claim or other claims.

Please apply any other charges or credits to deposit account 06-1050, referencing attorney docket 10559-886001.

Respectfully submitted,

Date: Oct. 12, 2006

Ido Rabinovitch
Attorney for Intel Corporation
Reg. No. L0080

PTO Customer No. 20985
Fish & Richardson P.C.
Telephone: (617) 542-5070
Facsimile: (617) 542-8906